Combining dynamic abstractions in very large MDPs

Kurt Steinkraus and Leslie Pack Kaelbling

What: The goal of our research is to develop a computer system capable of creating plans of action and then executing those plans in very large, stochastic domains.

Provable optimality is intractable for larger problems, so our goal is instead to create a system that has performance characteristics more like those of a human. The system attempts to find at least some solution, with the goal of finding a solution whose quality is "good enough." By emphasizing finding a good-enough plan rather than an optimal plan, the system's performance can degrade relatively gracefully as the problem size increases, similar to human problem-solving performance and in contrast to most previous methods for stochastic planning problems.

Our approach to planning in very large domains uses *dynamic abstraction modules* and combines modules into a *module hierarchy*. Modules operate successively on the domain, gradually transforming it into a highly abstract version. Each module knows how to exploit a particular type of structure in the domain, and its function is to exploit that structure, abstracting away any unnecessary details.

Each module provides a mapping describing its abstraction transformation, which allows the framework to plan and execute in the resulting highly abstracted domain, when in fact planning and execution in the module hierarchy end up being distributed across the modules.

Why: There are many existing methods for tackling very large stochastic planning problems, and each is useful at different times. A problem with most of these methods, though, is that they exploit only one type of structure. Thus these methods do not scale up well: they either don't simplify the problem enough to make it tractable, or they oversimplify the problem and lose important structure.

Another common problem with existing methods is that the representation is static. It is desirable to take advantage of the way that certain domain parts become more or less important in different situations, by planning over the whole domain in approximate terms and then filling in more detailed plans as corresponding domain parts are visited. That way, no wasted detailed planning is done for domain parts that are never visited. Rather than trying to modify a planner to exploit all the different ways of making an approximate plan, a better approach is to dynamically change the representation.

Plamnning with dynamic abstraction modules addresses both of these issues. Using many different types of modules allows the exploitation of many types of domain structure without accidentally discarding important information pertinent to building a good plan. Being able to change the detail level of the representation dynamically means that the plan's level of detail can be dynamically adjusted simply by keeping it in step with the representation.

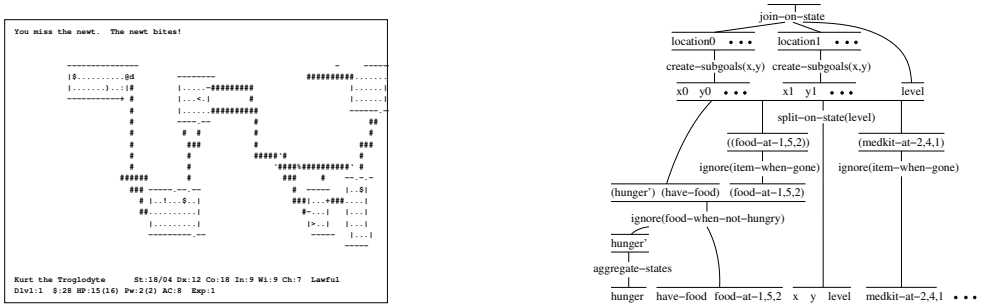

Figure 1: The module hierarchy used to solve the simplified nethack domain.

Progress: We have implemented the module hierarchy framework and several modules, testing its performance on a simplified version of the computer game `nethack` (`http://www.nethack.org/`). Nethack is a good domain for testing different approaches to solving real-world problems because it contains several different types of structure, some simple and some complex. The varying structure and the interaction between the different parts is representative of even larger, real-world domains, such as a disaster relief robot, a Mars rover, or a general purpose battlefield robot.

In the particular simplified version of nethack that we used, the goal is to escape from a dungeon, which has several levels with an escape stairway at the top. The game is not just a path planning problem, because the player has to stay well-fed and healthy, finding food and avoiding monsters. The module hierarchy that we created to solve this domain uses 18 modules instantiated from six module types. The module types implemented included modular adaptations of modified policy iteration [3], macro-actions [1], options [2], simple state aggregation, and selective state variable removal.

In comparing the running time and solution quality of the module hierarchy to policy iteration and to several abstraction methods operating individually, we ran experiments on a sequence of nethack domains of gradually increasing size. The running time results show that the module hierarchy is the only method that scales up to problems with very large numbers of states. The reward gained by all methods was equal (modulo randomness) on the instances where they succeeded in producing a solution.

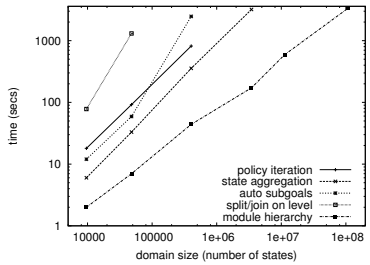

Figure 2: Experimental results.

Although the module hierarchy makes no guarantees about optimality, the nethack domain results show that, if modules are chosen and fitted together carefully, it may not be necessary to sacrifice much optimality in order to gain tractability. In fact, it is precisely because modules can be chosen to take advantage of the specific structure of different parts of the domain that larger instances of the nethack are tractable and yet the same level of reward is achieved.

Future: The execution times show that the module hierarchy can handle larger domains than any single abstraction method; even so, there is room for considerable improvement. For instance, most of the module hierarchy's time and memory is spent creating and caching abstract dynamics. If the transition distributions and reward functions were structured, e.g., as algebraic decision diagrams, then the abstract dynamics could be created much more quickly while still retaining their compact structure.

Other ways to improve on this first module hierarchy system include monitoring abstract action execution and interrupting when low probability occurrences can be exploited, measurement of the applicability and "fit" of modules to certain situations, and extension to other models like POMDPs and relational MDPs. An eventual goal is to create a black-box system, where the modules themselves to determine where and when they apply, automatically constructing and rearranging the module hierarchy.

Research Support: This work is being supported in part by NASA award #NCC2-1237 and in part by DARPA contract #DABT63-99-1-0012.

References:

[1] M. Hauskrecht, N. Meuleau, L. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of Markov decision processes using macro-actions. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 220–229. Morgan Kaufmann, 1998.

[2] D. Precup, R. Sutton, and S. Singh. Theoretical results on reinforcement learning with temporally abstract options. In *Proceedings of the Tenth European Conference on Machine Learning*, pages 382–393. Springer-Verlag, 1998.

[3] M. Puterman and M. Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137, 1978.